Chapter 4
Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms

Irina Dumitrescu and Thomas Stützle

Abstract Exact mathematical programming techniques such as branch-and-bound or dynamic programming and stochastic local search techniques have traditionally been seen as being two general but distinct approaches for the effective solution of combinatorial optimization problems, each having particular advantages and disadvantages. In several research efforts true hybrid algorithms, which exploit ideas from both fields, have been proposed. In this chapter we review some of the main ideas of several such combinations and illustrate them with examples from the literature. Our focus here is on algorithms that have the main framework given by the local search and use exact algorithms to solve subproblems.

4.1 Introduction

Many problems arising in areas such as scheduling and production planning, location and distribution management, Internet routing or bioinformatics are combinatorial optimization problems (COPs). COPs are intriguing because they are often easy to state but often very difficult to solve, which is captured by the fact that many of them are $\mathcal{NP}$-hard [48]. This difficulty and, at the same time, their enormous practical importance, have led to a large number of solution techniques for them. The available solution techniques can be classified as being either *exact* or *approximate* algorithms. Exact algorithms are guaranteed to find an optimal solution and *prove* its optimality for every

Irina Dumitrescu
School of Mathematics, University of New South Wales, Sydney, Australia
e-mail: `irina.dumitrescu@unsw.edu.au`

Thomas Stützle
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium
e-mail: `stuetzle@ulb.ac.be`

finite size instance of a COP within an instance-dependent, finite run-time, or prove that no feasible solution exists. If optimal solutions cannot be computed efficiently in practice, it is usual to trade the guarantee of optimality for efficiency. In other words, the guarantee of finding optimal solutions is sacrificed for the sake of getting very good solutions in reasonably short time by using approximate algorithms.

Two solution method classes that have significant success are *integer programming* (IP) methods as an exact approach, and *stochastic local search* (SLS) algorithms [61] as an approximate approach. IP methods rely on the characteristic of the decision variables being integers. Some well known IP methods are branch-and-bound, branch-and-cut, branch-and-price, and dynamic programming [86]. In recent years, remarkable improvements have been reported for IP methods when applied to some problems (see, e.g. [11] for the TSP). Important advantages of exact methods for IP are that (i) *proven* optimal solutions can be obtained if the algorithm succeeds, (ii) valuable information on upper/lower bounds on the optimal solution are obtained even if the algorithm is stopped before completion (IP methods can become approximate if we define a criterion of stopping them before solving the problem), and (iii) IP methods allow to provably prune parts of the search space in which optimal solutions cannot be located. A more practical advantage of IP methods is that research code such as Minto [85] or GLPK [54], or powerful, general-purpose commercial tools such as CPLEX [63] or Xpress-MP [113] are available. However, despite the known successes, exact methods have a number of disadvantages. Firstly, for many problems the size of the instances that are practically solvable is rather limited and, even if an application is feasible, the variance of the computation times is typically very large when applied to different instances of a same size. Secondly, the memory consumption of exact algorithms can be very large and lead to the early abortion of a program. Thirdly, for many COPs the best performing algorithms are problem specific and they require large development times by experts in integer programming. Finally, high performing exact algorithms for one problem are often difficult to extend if some details of the problem formulation change. The state-of-the art for exact algorithms is that for some $\mathcal{NP}$-hard problems very large instances can be solved fast, while for other problems even small size instances are out of reach.

SLS is probably the most successful class of approximate algorithms. When applied to hard COPs, local search yields high-quality solutions by iteratively applying small modifications (local moves) to a solution in the hope of finding a better one. Embedded into higher-level guidance mechanisms, which are called (general-purpose) SLS methods [61]or, more commonly, metaheuristics, this approach has been shown to be very successful in achieving near-optimal (and often optimal) solutions to a number of difficult problems [1, 61, 108]. Examples of well-known general-purpose SLS methods (or metaheuristics) are simulated annealing, tabu search, memetic algorithms, ant colony optimization or iterated local search [52]. Advantages of SLS algorithms are that (i) they are the best performing algorithms available for a variety of problems, (ii) they can examine a huge number of possible solutions in short computation time, (iii) they are often more easily adapted to slight variants of problems and are therefore more flexible, and (iv) they are typically easier to understand and implement by the common user than exact methods. However, local search based algorithms have several disadvantages. Firstly, they cannot prove optimality and typically do not give bounds on the quality of the solutions they return. Secondly, they typically cannot *provably* reduce the search space. Thirdly, they do not have well defined stopping criteria (this is particularly true for metaheuristics). Finally, local search methods often have problems with highly constrained problems where feasible areas of the solution space are disconnected. Another problem that occurs in practice is that, with very few exceptions [110], there are no efficient general-purpose local search solvers available. Hence, although one can typically develop an SLS algorithms of reasonable performance rather quickly, many applications of SLS algorithms can require considerable development and implementation efforts if very high performance is required.

It should be clear by now that IP and SLS approaches have their particular advantages and disadvantages and can be seen as complementary. Therefore, it appears to be a good idea to try to combine these two distinct techniques into more powerful algorithms. Many articles use the probably most straightforward of these combinations, which is the use of an SLS algorithm to compute good intial upper bounds (in the minimization case) that allow an early pruning of non-optimal solutions. Intuitively, one would expect more advanced combinations of exact and local search methods to result in even stronger algorithms.

In this chapter we focus on approaches that strive for an integration of IP and SLS techniques. In particular, we describe ways of integrating IP methods (or methods derived from an IP approach) into SLS methods. In other words, we focus on approaches where the main algorithm (the master) is the local search technique and the IP method is typically used to solve some subproblems. For an overview of methods where as the master method can be considered the IP method and SLS techniques are used to solve subproblems, we refer to Chapter 3 of this book. We refer to this chapter and Chapter 2 for methods that use the spirit of local search inside IP methods (such as local branching [46]).

More in detail, we will cover the following five ideas that can be abstracted from the various true hybrid algorithms proposed in the literature: (i) the usage of exact algorithms to explore large neighborhoods in SLS algorithms, (ii) ways of enhancing metaheuristics by solving some subproblems exactly, (iii) the usage of techniques taken from branch-and-bound to enhance constructive SLS methods, (iv) the exploitation of the structure of good solutions identified by SLS algorithms, and (v) the exploitation of information from relaxations in SLS algorithms. In fact, the first three ways of combining local search and exact algorithms are classified in [96] as integrative combinations, while the last two are classified as collaborative combinations: the two methods are executed in sequence. For each of these five groups, we will describe some general ideas, give one or more detailed examples, and give pointers to literature for other ideas following the corresponding idea.

Before proceeding with our discussion we need to clarify the notion of *solution component* and *decision variable* that we will use in our chapter. We explain what we mean by using the example of the symmetric traveling salesman problem (TSP). The TSP can be defined on an undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges. An edge represents a pair of nodes between which travel is possible. Every edge $\{i, j\}$ has an associated cost d_{ij} and the goal in the TSP is to find a Hamiltonian circuit of minimal length. When applying local search to the TSP, a solution component usually refers to a single edge in a tour; each tour contains exactly n such solution components. More generally, a solution component can be seen as an atomic item and a set of such atomic items defines a problem solution. The analogue to a solution component in IP formulations is the notion of decision variable. For example, in the usual IP formulations of the TSP (see [86] for an overview), a variable $x_{ij} \in \{0, 1\}$ is associated with each edge $\{i, j\}$. The variable x_{ij} is equal to 1, if the edge $\{i, j\}$ is included in a tour and zero, otherwise. The x_{ij} variables are called decision variables. In the following sections, if we say that a decision variable is fixed to one (zero), we mean that we consider only those solutions in which this variable is set to one (zero). Analogously, in the local search case we say that a solution component is forced to always occur (not occur) in any solution considered. In the case of the TSP, this corresponds to an edge always (never) being used in any of the solutions considered. If a variable is free, it can take any value in $\{0, 1\}$.

4.2 Exploring large neighborhoods

Given a current solution s, (perturbative) local search [61] explores the neighborhood $\mathcal{N}(s)$ of s iteratively and tries to replace s by a solution $s' \in \mathcal{N}(s)$ according to some criterion. In local search, it is appealing to search large neighborhoods because much better solutions can be reached in one local search step than when using simple, small neighborhoods. However, large neighborhoods have the associated disadvantage that a considerable amount of time may be spent to search them in order to find some or the best improving neighboring solution. More than that, many of the large neighborhoods proposed in the literature are exponentially large with respect to instance size [51, 70, 93, 106, 107] and the main potential bottleneck for local search in large neighborhoods is the task of searching for a better or the best solution in the neighborhood of the current one.

Local search algorithms for large neighborhoods can be classified roughly into two classes. The first class comprises those where the large neighborhood is searched heuristically in some problem-specific way. Examples of this class of local search algorithms are variable-depth search algorithms [70, 65] or ejection chains [51]. The second are local search algorithms where the neighborhood is searched exactly, i.e. the central idea of these algorithms is to model the problem of searching a large neighborhood as an optimization problem, which is solved exactly. The solution of the resulting search problem will determine the neighbor that will replace the current solution in the local search.

Two main possibilities are considered for defining the neighborhood search problem. The first one is to define a special-purpose neighborhood and to model the exploration of the *full* neighborhood as an optimization problem. In this case, a search problem is defined such that each feasible solution of it induces a move of the local search algorithm. Clearly, the task of the exact algorithm in this case will be to solve the resulting *neighborhood search problem* (NSP). A general algorithmic outline of this idea can be given as follows:

Algorithm 4.2.1 *Neighborhood search*

```
Step 1: Initialization
        Let $s$ be a feasible initial solution.
Step 2: Local search
        while stopping_criterion is not met do
            Define a search problem $\mathcal{P}(s)$ that depends on $s$.
            Find $opt(\mathcal{P}(s))$, an optimal solution of $\mathcal{P}(s)$.
            Let $s'$ be the solution induced by $opt(\mathcal{P}(s))$.
            if $s'$ is better than $s$ w.r.t. the objective function then $s = s'$.
        enddo
Step 3: Return $s$.
```

The stopping criterion may be triggered, e.g., if no improved solution s can be obtained or the solutions to $\mathcal{P}(s)$ are infeasible etc. The first two examples that we present, very large scale neighborhood search and Dynasearch, fall in this class of NSP hybrids. Typically, algorithms that are based on NSP try to determine composite moves, which are composed of a sequence or set of simpler moves.

The second “possibility” is that at each step of the local search a part of the current solution s is maintained fixed, thus defining a partial solution, while the values of the rest of the decision variables are left free. The free part is then rearranged optimally, subject to the constraint that the fixed part cannot be altered. In this sense, the task of the exact algorithm is to solve the *partial neighborhood search problem* (PNSP). The following algorithmic outline gives a high level view of such a procedure in a first-improvement style of local search. The outline sketches an iterative improvement algorithm that stops in the first locally optimal solution encountered. The operator $\otimes$, which is used below, joins two partial solutions into a complete solution. A distinctive feature to the NSP type algorithms is that the final move is not considered to be composed of simple underlying moves.

Algorithm 4.2.2 *Partial neighborhood search*

```
Step 1: Initialization
        Let $s$ be an initial solution.
Step 2: Neighborhood search
        while improvement found do
Step 3: Neighborhood scan
            while not full neighborhood examined do
                Delete solution components from solution $s$ resulting
                    in a partial solution: $s_p = s \setminus r$.
                Define a search problem $\mathcal{P}(r)$.
                Find $opt(\mathcal{P}(r))$, the optimal solution of $\mathcal{P}(r)$.
                Perform the move induced by $opt(\mathcal{P}(r))$, resulting in
                solution $\bar{s}'$; $s' = s_p \otimes \bar{s}'$.
                if $s'$ is better than $s$ w.r.t. the objective function then $s = s'$.
            enddo
        enddo
Step 4: Return $s$.
```

We illustrate the partial neighborhood search principle on the hyperopt local search algorithm. In fact, hyperopt is only one example of such methods. They can be traced back to Applegate and Cook's shuffle heuristic [12].

4.2.1 NSP Example: Cyclic and Path Exchange Neighborhoods

One application of NSP is the very large scale neighborhood (VLSN) search that was introduced by Ahuja et al. [5, 6] in the context of solving partitioning problems. Examples of partitioning problems include graph coloring, vehicle routing, generalized assignment, parallel machine scheduling, clustering, aggregation, and graph partitioning. Next, we briefly describe the type of problems to which VLSN search is applied and how the neighborhood search problem is defined.

A partition $\mathcal{T}$ of a set W of n elements is a set of subsets $\mathcal{T} = \{T_1, \ldots, T_K\}$ such that $W = T_1 \cup \cdots \cup T_K$ and $T_k \cap T_{k'} = \emptyset$, $k, k' = 1, \ldots, K$. To each set T_k one can associate a cost $c_k(T_k)$. Then, the partitioning problem is to search for a partition of W such that the sum of the costs of the partitions is minimal. The properties of the cost function are not important for now, except that it is separable over subsets.

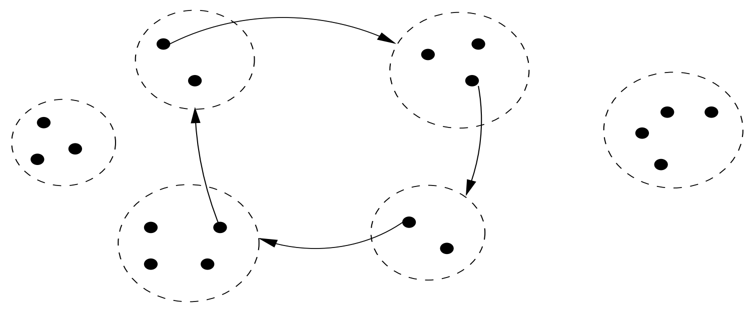

Fig. 4.1 Example of a cyclic exchange for a partitioning problem.

The one-exchange neighborhood, which is based on the move of one element of W from one partition into another one, and the two-exchange neighborhood, which is based on exchanges of two elements of two different partitions, are typical simple neighborhoods for partitioning problems. Cyclic exchange neighborhoods generalize the two-exchange neighborhood [6, 106, 107] by exchanging not only two elements from two different subsets but moving several elements, each of a different subset. An illustration of this neighborhood is given in Figure 4.1. A path exchange is similar to a cyclic exchange except that from the last subset of the path no element is removed.

The cyclic (path) exchange neighborhood of a partition $\mathcal{T}$ is then the set of all partitions $\mathcal{T}'$ that can be obtained by a cyclic (path) exchange as described above. In what follows we focus on the cyclic exchange neighborhood. A cyclic exchange modifies the cost of the partitions involved in the move. If the cost function is separable over subsets, the cost of a cyclic exchange can be obtained as the sum of the cost differences of all subsets that are modified by the move.

Ahuja et al. define an *improvement graph* on which a search problem is solved in order to find an improving cyclic exchange. The node set V of the improvement graph is defined as the collection of integers $1, \ldots, n$ such that each node in V is in one-to-one correspondence to an element of W. For each pair of elements that do not belong to the same subset in $\mathcal{T}$, an arc (i, j) between the corresponding nodes is added in the improvement graph if and only if the subset to which the element corresponding to node j belongs remains feasible after the removal of the element corresponding to j and the addition of the element corresponding to i. The partition $\mathcal{U}$ of V is the collection of subsets $U_1, \ldots, U_K$ that corresponds to the subsets in $\mathcal{T}$. Hence, we have that a cyclic exchange in $\mathcal{T}$ corresponds to a subset disjoint cycle in the improvement graph with respect to $\mathcal{U}$. The cost associated to any arc (i, j) is defined to be equal to the difference between the cost of the set after the removal of the element corresponding to j and the addition of the element corresponding to i, and the cost of the original set that contains the element corresponding to j. Thompson and Orlin [106] showed that there is a one-to-one correspondence between the cyclic exchanges with respect to $\mathcal{T}$ and the subset-disjoint cycles in the improvement graph (with respect to $\mathcal{U}$) and that both have the same cost.

The search for a best neighboring solution can then be modeled as a search problem on the improvement graph. In fact, for determining an improving cyclic exchange move, a subset disjoint negative cost cycle (SDNCC) needs to be found in the improvement graph, i.e. a cycle that uses at most one node from every subset of the partition of the set of nodes. Determining the best among the improving neighbors corresponds to finding the most negative such cylce (this problem is called the SDNCCP) and needs to be solved exactly. Several exact methods for the SDNCCP are proposed by Dumitrescu [40]. She also proposes algorithms for the more general problem of determining a subset disjoint minimum cost cycle (SDMCC), which includes the case when such a minimum cost cycle need not be negative (i.e., it does not correspond to an improvement). Such cases are important if, e.g. the VLSN technique is embedded into a tabu search. The algorithms for determining the optimal solutions to SDNCCP and SDMCCP can be seen as generalizations of dynamic programming algorithms for shortest path problems. An acceleration method that exploits symmetries for the SDMCCP is given, and an elegant theorem of Lin and Kernighan [70] is exploited in the SDNCCP case. Although both problems are $\mathcal{NP}$-hard, the proposed algorithms have been shown to be very efficient for the subproblems that arise in practical applications of VLSN search methods. Dumitrescu also proposes heuristics that are obtained by limiting or truncating the exact methods in some way.

An exact solution of SDNCCP was also presented by Ahuja et al. [7]. They integrated the exact solution of the SDNCCP into a VLSN local search for the capacitated minimum spanning tree problem and obtained very good results, improving the best known solutions for 36% of the tested benchmark instances. A more recent study of an exact evaluation of cyclic and path exchange neighborhoods for graph coloring has been presented in [33]. Although the usage of these neighborhoods gave significant improvements with respect to the solution quality reachable for iterative improvement algorithms for graph coloring [33], once these neighborhoods have been integrated into metaheuristic algorithms, the overhead in computation time made the approach undesirable. Since this may happen also for other problems, it is not surprising that the exact solution of the neighborhood search problem in cyclic and path exchange neighborhoods is sometimes replaced by a heuristic solution to the SDNCCP or SDMCCP, which are, however, often derived from a possible exact solution of the neighborhood search problem.

4.2.2 NSP Example: Dynasearch

Dynasearch is a local search technique that uses a dynamic programming algorithm to search for the best possible combination of mutually independent, simple search moves to be executed in one local search iteration. In the context of dynasearch, the independence of the simple moves refers to the requirements that they do not interfere with each other with respect to (i) the evaluation function being used and (ii) the feasibility of a solution after the execution of the moves. In particular, the independence of the moves with respect to the evaluation function implies that the gain obtained by a dynasearch move can be obtained as the sum of the gains of the simple moves.

So far, various applications of dynasearch to permutation problems have been proposed. The independence between simple search moves is typically related to the fact that the indices affected by the simple moves are not overlapping. Let $\pi = (\pi(1), \ldots, \pi(n))$ be the current permutation. Two moves that involve elements from $\pi(i)$ to $\pi(j)$ and $\pi(k)$ to $\pi(l)$, with $1 \leq i < j \leq n$ and $1 \leq k < l \leq n$ need to have that either $j < k$ or $l < i$.

The best combination of independent moves can be obtained by a dynamic programming algorithm, which gives the name of this technique. Let $\Delta(j)$ be the maximum total cost reduction obtained from independent moves affecting only positions 1 to j of the current permutation and let $\delta(i, j)$ be the cost reduction by a move involving positions between i and j, including i and j. The idea of the algorithm is to compute the maximum total cost reduction obtained by either appending element $\pi(j)$ to the partial permutation of the elements $\pi(1)$ to $\pi(j-1)$ or by appending element $\pi(j)$ and applying a move involving element $\pi(j)$ (and possibly elements from $\pi(i)$ onwards).

The values of $\Delta(j), j = 0, \ldots, 1$ can be computed in a dynamic programming fashion. Let π be the current solution and set $\Delta(0) = 0$ and $\Delta(1) = 0$. Then, $\Delta(j),\ j = 1, \ldots, n-1$ can be obtained in a forward evaluation by the recursive formula

$$\Delta(j+1) = \max\{\max_{1 \leq i \leq j}\{\Delta(i-1) + \delta(i, j+1)\}, \Delta(j)\}. \tag{4.1}$$

$\Delta(n)$ gives the largest reduction in solution cost. The compound move to be executed can then be determined by tracing back the computation steps.

When applying the algorithm, the particularities of the problem and of the moves may have to be taken into account to slightly adapt the dynamic programming recursion given by Equation (4.1). (Note that the dynasearch algorithm can be cast in terms of a search for a shortest path in an appropriately defined improvement graph [3].)

Current applications of dynasearch comprise the TSP [34], the single machine total weighted tardiness problem (SMTWTP) [34, 35, 55] and a time-dependent variant of it [8], and the linear ordering problem (LOP) [34]. A general observation is that dynasearch, on average, is faster than a standard best-improvement descent algorithm and may return slightly better qual-

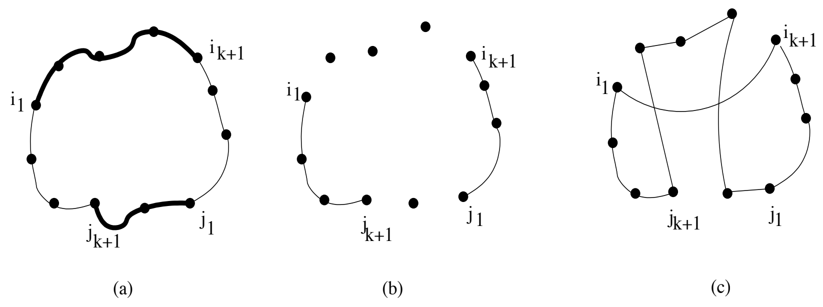

Fig. 4.2 (a) A feasible tour can be seen as the union of $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, and $\mathcal{H}(j_{k+1}, i_1)$. (b) The hyperedges $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ are removed. (c) A new feasible tour is constructed.

ity solutions. Particularly good performance is reported when dynasearch is used as a local search routine inside other metaheuristics such as iterated local search [73]. Currently, iterated dynasearch is the state-of-the-art method for SMTWTP [55], and very good results were obtained for the TSP and the LOP [34].

4.2.3 PNSP Example: Hyperopt Neighborhoods

A relatively simple example of a PNSP approach is the hyperopt local search approach applied to the TSP [27, 37, 28]. To keep the presentation as simple as possible, we will only consider its application to the symmetric TSP.

The *hyperopt neighborhood* is based on the notion of hyperedges. Given a tour of the TSP, a *hyperedge* is defined to be a subpath of the tour; in other words, a sequence of successive edges of the tour [37]. If i is the start node and j the end node of the hyperedge, we denote the hyperedge by $\mathcal{H}(i, j)$. The *length* of a hyperedge is given by the number of edges in it. Let t be a feasible tour of the TSP and $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ two hyperedges of length k such that $\mathcal{H}(i_1, i_{k+1}) \cap \mathcal{H}(j_1, j_{k+1}) = \emptyset$ with respect to the nodes contained. We assume that the tour t can be written as $t = (i_1, \ldots, i_{k+1}, \ldots, j_1, \ldots, j_{k+1}, \ldots, i_1)$. It is obvious that the tour t can be described completely by four hyperedges: $\mathcal{H}(i_1, i_{k+1})$, $\mathcal{H}(i_{k+1}, j_1)$, $\mathcal{H}(j_1, j_{k+1})$, and $\mathcal{H}(j_{k+1}, i_1)$, as shown in Figure 4.2(a). A *k-hyperopt move* is defined as a composition of the two following steps: remove $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ from the tour t, then add edges to $\mathcal{H}(i_{k+1}, j_1)$ and $\mathcal{H}(j_{k+1}, i_1)$ such that a new feasible tour is constructed (see Figure 4.2 for an example). The *k-hyperopt neighborhood* consists of all k-hyperopt moves.

The size of the k-hyperopt neighborhood increases exponentially with k. The authors propose an “optimal” construction of a k-hyperopt move by solving exactly a subproblem: the TSP defined on the graph $G' = (V', E')$, where V' is the set of nodes included in $\mathcal{H}(i_1, i_{k+1})$ and $\mathcal{H}(j_1, j_{k+1})$ and E' is the set of edges in the original TSP that have both ends in V'. However, this approach is bound to be efficient only when k is relatively small. Otherwise, a large size TSP would have to be solved as a subproblem. To avoid this, the authors propose to use a dynamic programming algorithm for the case $k \geq 3$ [28, 37], but no details of the algorithm are given.

For a given hyperedge $\mathcal{H}(i_1, i_{k+1})$ the k-hyperopt move to be performed is determined in [28, 37] in a first descent form. Hence, a sequence of subproblems is generated and solved, one subproblem for every hyperedge that does not intersect with $\mathcal{H}(i_1, i_{k+1})$. To speed up computations, usual TSP speed-up techniques such as "don't look bits" are used [19]. In addition, extensions based on embeddings of the hyperopt local search into iterated local search and variable neighborhood search algorithms are considered. However, the currently available results suggest that the resulting hyperopt local search for the ATSP is somewhat inferior to current state-of-the-art local search techniques [64]. In spite of this, the k-hyperopt approach may have potential if the size of the neighborhood is greatly enlarged and truly efficient algorithms such as Concorde [10] are used to solve the resulting subproblems.

4.2.4 Other Approaches

The two NSP techniques for exploring a neighborhood that we have described in this section have been applied to a variety of problems. For a recent overview of VLSN search applications we refer to the book chapter by Ahuja et al. [4]. Recently, the corridor method has been proposed by Sniedovich and Voß [103]. The central idea put forward is to define constraints on the problems, such that efficient exact methods can be designed to solve the neighborhood search problem efficiently. Another example of an NSP method is the constraint programming approach by Pesant and Gendreau [89, 90], where the exploration of the neighborhood is modeled as a problem that is then solved with constraint programming techniques [78, 109]. Constraint programming is an especially promising approach, if the resulting search problems are tightly constrained.

PNSP algorithms have been more widely applied. As mentioned before, one of the first PNSP algorithms is the shuffle heuristic of Applegate and Cook [12], which was applied to the job-shop scheduling problem. Briefly, their approach consists of defining a partial solution by leaving the sequence fixed for one or a few machines. The subproblem of completing the schedule is then solved by branch-and-bound. Another example is the MIMAUSA algorithm, introduced by Mautor and Michelon [79, 80, 81], applied to the quadratic assignment problem. In MIMAUSA, at each step k variables are "freed". The subproblem of assigning values to the free variables, such that the rest of the solution is kept fixed, is solved to optimality. A similar approach has been applied by Büdenbender et al. [26] to the direct flight network design problem. Hu et al. [62] search large neighborhoods for the generalized minimum spanning tree problem. In particular, they implemented a variable neighborhood descent algorithm, where in the largest neighborhood a mixed-integer program (MIP) solver is used to determine optimal solutions to the subproblems arising in the local search. Computational results show that their approach obtained excellent performance. A similar approach of searching large neighborhoods by a MIP solver is applied by Prandtstetter and Raidl to a car sequencing problem [94]. PNSP methods have been successfully applied in combination with constraint programming techniques. Some noteworthy examples are the work of Caseau and Laburthe [31] for the job-shop scheduling problem or that of Shaw [102] for the vehicle routing problem, where a branch-and-bound algorithm is integrated into tabu search. An earlier overview of combinations of local search and constraint programming is given in an article by Focacci et al. [47].

4.2.5 Discussion

In general, the use of exact algorithms for exploring large neighborhoods is appealing. For some problems the resulting neighborhoods can be explored fully by polynomial time algorithms, a noteworthy example being here the dynasearch approach. If the large neighborhoods cannot be explored in *provably* polynomial time, often the resulting neighborhood search problems, whether corresponding to full or partial neighborhood, can be solved efficiently by exact algorithms. This is mainly due to the exact algorithms being, in most cases, rather quick if the problem size is not too large and to the fact that the resulting subproblems often have a special structure that can be exploited efficiently.

Despite the usefulness of exact algorithms in these methods, it is clear that the resulting subproblems could be solved by approximate algorithms. In fact, this is often done. A general framework for this idea of defining subproblems and exploring them heuristically, POPMUSIC, was defined in a paper of Taillard and Voß [104]. Similar ideas were put forward in variable neighborhood decomposition search [58]. It would be interesting to study more carefully the use of heuristics versus the use of exact algorithms for exploring the large neighborhoods using sound experimental designs and try to derive more general insights into when one is prefered of the other.

4.3 Enhancing Metaheuristics

Occasionally, exact methods are employed to implement central sub-procedures in a metaheuristic, typically with the goal of adding specific features of diversification or intensification of the search that are beyond the reach of local search algorithms. One early example of such a combination is the usage of exact algorithms as perturbation operators in iterated local search.

4.3.1 Example: Perturbation in Iterated Local Search

A successful approach to solving difficult COPs is iterated local search (ILS) [73]. ILS consists of running a local search procedure, starting from solutions obtained by perturbing available local optima. A simple form of an ILS is given in Algorithm 4.3.1.

Algorithm 4.3.1 *Iterated Local Search*

```
Step 1: Let $s$ be an initial solution.
Step 2: while stopping_criterion is not met do
(i)          Let $s'$ be the solution obtained from $s$ after a perturbation.
(ii)         Call local_search($s'$) to produce the solution $s''$.
(iii)        if $s''$ is accepted as the new incumbent solution then $s = s''$.
        enddo
Step 3: Return $s$.
```

The main role of the perturbation in ILS is to introduce significant changes in the current solution in order to allow the local search to explore different local optima, while still conserving good characteristics of a current solution. Simple ways of introducing such perturbations, like applying a random move in a large neighborhood, are usually enough to obtain good performance. However, state-of-the-art performance is often reached by more problem specific perturbations and when the perturbations introduce some structural changes that cannot be reversed easily by a local search [73].

One possibility of combining exact algorithms with ILS is to let an exact algorithm determine the perturbation. This may be very useful, since the exact algorithm may introduce larger structural changes, which cannot be easily undone by the local search. The implementation of this idea can be done by first fixing some part of the solution and leaving the rest free. Next, the free part is optimized, and the solution to the free part is reinserted into the fixed part, possibly after restoring feasibility if this should be necessary. In other words, at Step 2(i) of Algorithm 4.3.1 a subproblem is solved to optimality. Formally, Step 2(i) is replaced by:

Algorithm 4.3.2 *Solving a subproblem to determine the perturbation (new Step 2(i) of ILS)*

```
begin
    Let $s'' = s \setminus r$, where $r \subset s$.
    Define $\mathcal{P}(r)$, a subproblem that depends on $r$.
    Let $opt(\mathcal{P}(r))$ be the optimal solution of $\mathcal{P}(r)$.
    Let $\bar{s'} = s'' \cup opt(\mathcal{P}(r))$.
    Modify $\bar{s'}$ such that feasibility is restored.
    Return $s'$, be the modified feasible solution.
end
```

An early example of how to determine a perturbation by solving a subproblem exactly is given by Lourenço [72]. In her paper, Lourenço conducts an extensive computational study of ILS applied to the job-shop scheduling problem (JSP). The JSP is defined for m machines and n jobs. Each job consists of a sequence of operations that have to be performed in a given order. Each operation has to be executed for a specified, uninterrupted time (i.e. preemption is not allowed). In the JSP, precedence constraints induce a total order on the operations of each job. Additional constraints require that each machine handles at most one job at a time. A feasible schedule is a schedule that satisfies all the precedence and capacity constraints. The JSP consists in finding a feasible schedule that minimizes the overall job completion time.

The JSP can be modeled using the disjunctive graph $G = (V, A, E)$ representation, where V is the vertex set corresponding to the operations, A is the arc set corresponding to the job precedence constraints, and E is the edge set corresponding to the machine capacity constraints [101]. In this graph, the scheduling decision corresponds to orienting each edge in E.

Lourenço experimentally tested several ways of defining the perturbation. One perturbation procedure proposed by Lourenço is making use of Carlier's algorithm [30], which is a branch-and-bound method applied to a one-machine scheduling problem. The particular problem solved here can be seen as a very simple version of the JSP: a number of operations need to be scheduled on one machine in the presence of temporal constraints. Lourenço's idea is to modify the directed graph corresponding to the current solution of the JSP by removing all the directions given to the edges associated with two randomly chosen machines. Then Carlier's algorithm is applied to one of the machines. The problem to solve is therefore a one-machine scheduling problem. Next, the edges corresponding to that machine are oriented according to the optimal solution obtained. Finally, the same treatment is applied to the second machine. Lourenço mentions that this perturbation idea can create cycles in the graph and suggests a way of obtaining a feasible schedule from the graph with cycles (see [72] for details). In conclusion, at each iteration of the ILS, two subproblems are solved in order to construct a new initial solution for the local search procedure. Lourenço proposed similar perturbation schemes for her ILS algorithm for the JSP; for details we refer to [71, 72].

Finally, it should be said that the computational results of Lourenço cannot be considered state-of-the-art performance. The main reason probably is that she used a rather weak local search algorithm when compared to the effective tabu search algorithms proposed by Nowicki and Smutnicki [87] or the local search procedure of Balas and Vazacopoulos [15]. However, it would be interesting to see how the performance of these two algorithms would be affected by the addition of the perturbation steps proposed by Lourenço.

4.3.2 Other Approaches

Exact algorithms can be used within particular metaheuristics for specific operations that are to be done while searching for solutions. An example in a somewhat similar spirit as the previously described one is followed in the recent paper by Fernandes and Lourenço [43]. They introduce occasional large perturbations into a tabu search algorithm, which consists in a partial destruction of a solution and a reconstruction of a new one. The reconstruction is guided by using violated valid inequalities that enforce some specific order among unscheduled operations. Apart from the perturbations in ILS, other examples can be found in applications of genetic algorithms. Here, exact algorithms are typically used in the recombination operation, in which two solutions s and s' are combined to form one or several new solutions (also called offspring). The main idea is to define a subproblem $\texttt{Recombination}(s, s')$ that comprises all the solutions that can result following a particular way of combining s and s', and then to search for the best solution of the resulting subproblem.

The subproblem can be obtained by keeping common solution features of the two "parent" solutions fixed and to try to find the best solution for the free parts. A second possibility is to define a subproblem consisting of the union of the solution components contained in the two parent solutions s and s'. An example for the first approach is presented by Yagiura and Ibaraki [114]. They apply this idea to permutation problems using a dynamic programming algorithm for finding an optimal permutation subject to the constraint that a partial order common to the two parent solutions s and s' is maintained by the new solution. Other examples include the MIP-recombination operator for a specific supply management problem [24] and for the balancing transfer lines problem described in Chapter 7. There, many of the variables that have the same value in the parents are fixed and the resulting subproblem is solved by a mixed-integer programming solver. The complexity of optimal recombination when solutions are represented as bitstrings is studied in [42]. Examples of the second idea are the papers of Balas and Niehaus [14] on the maximum clique problem (MCP), and of Aggarwal et al. [2] on the maximum independent set problem (MISP). The MCP and MISP are subset problems in graphs, where a subset of the vertices needs to be chosen. Hence, in both cases, a solution corresponds to a subset of the vertices of the original graph $G = (V, A)$, i.e. we have $s, s' \subset V$. In both cases, the subproblem consists of a subgraph comprising all the vertices in $s \cup s'$ and all edges that connect them. It can be shown that the resulting subproblem is a matching problem in a bipartite graph. Hence, it can be solved in polynomial time [2, 13]. Another paper following these lines is that of Lourenço et al. [74] on a multi-objective bus driver scheduling problem, where analogous ideas are applied to the offspring determination of a genetic algorithm applied to a set covering problem.

4.3.3 Discussion

Clearly, the techniques discussed here are specific to particular metaheuristics. However, these examples indicate possible areas of interest, where such combinations can be useful. In general, mainly hybrid metaheuristics will be candidates for possible combinations. We call hybrid metaheuristics those metaheuristics that consist of the combination of several clearly distinct procedures like perturbation and local search in the ILS case or recombination, mutation, and (possibly) local search in the genetic algorithm case. In fact, the examples outlined above illustrate well the potential of such combinations. Other metaheuristics that could profit from such combinations are ant colony optimization [38], e.g. by optimizing partial solutions or extending partial solutions in an optimal way, or scatter search [53], in a way analogous to what is described for the genetic algorithms.

4.4 Using Branch-and-Bound Techniques in Constructive Search Heuristics

Construction heuristics build solutions starting from an empty partial solution. They iteratively add solution components until a complete solution is obtained. Construction heuristics typically rate the desirability of adding a solution component based on a heuristic measure of its objective function contribution. Then, they either add a solution component greedily, i.e. by choosing the best rated component, or probabilistically, biased by the heuristic information.

In construction heuristics partial solutions are typically only extended and never reduced. However, construction heuristics can easily by transformed into tree search algorithms, e.g. by adding a backtracking-type mechanism and usages of lower and upper bounds. This finally leads to branch-and-bound algorithms [61].Hence, a somewhat natural hybrid is to extend construction heuristics or metaheuristics that are based on the repeated construction of solutions such as ant colony optimization [39] by tree search techniques.

4.4.1 Example: Approximate Nondeterministic Tree Search (ANTS)

Ant Colony Optimization (ACO) [39] is a recent metaheuristic approach for solving hard COPs. In ACO, (artificial) ants are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions while taking into account (i) heuristic information on the problem instance being solved, if available, and (ii) (artificial) pheromone trails. Pheromone trails in ACO serve as distributed, numerical information which is adapted during the execution of the algorithm to reflect the search experience. Of the available ACO algorithms, the approximate nondeterministic tree search (ANTS) algorithm [75] was the first to integrate branch-and-bound techniques into ACO.

ANTS was first applied to the quadratic assignment problem (QAP). In the QAP, a set of objects has to be assigned to a set of locations with given distances between the locations and given flows between the objects. The goal in the QAP is to place the objects on locations in such a way that the sum of the product between flows and distances is minimal. More formally, in the QAP one is given n objects and n locations, values a_{ij} representing the distance between locations i and j, and b_{rs} representing the flow between objects r and s. Let x_{ij} be a binary variable which takes value 1 if object i is assigned to location j, and 0 otherwise. The problem can be formulated as follows:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} a_{ij} b_{kl} x_{ik} x_{jl} \tag{4.2}$$

subject to the constraints

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{4.3}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{4.4}$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1, \ldots, n \tag{4.5}$$

In ANTS, each ant constructs a solution by iteratively assigning objects to a free location. Hence, the solution components to be considered are pairings between objects and locations (there are n^2 such solution components). Given a location j, an ant decides to assign object i to this location with a probability given by

$$p_{ij}^{k}(t) = \frac{\alpha \cdot \tau_{ij}(t) + (1-\alpha) \cdot \eta_{ij}}{\sum_{l \in \mathcal{N}_j^k} \alpha \cdot \tau_{lj}(t) + (1-\alpha) \cdot \eta_{lj}} \qquad \text{if } i \in \mathcal{N}_j. \tag{4.6}$$

Here, $\tau_{ij}(t)$ is the pheromone trail associated to the assignment of object i to a location j, which gives the "learned" desirability of choosing this assignment (pheromones vary at run-time), η_{ij} is the heuristic desirability of this assignment, α is a weighting factor between pheromone and heuristic, and $\mathcal{N}_j$ is the feasible neighborhood, i.e. the set of objects that are not yet assigned to a location.

Lower bound computations are exploited at various places in ANTS. Before starting the solution process, ANTS first computes the Gilmore-Lawler lower bound (GLB) [49, 68]. The GLB is defined to be the solution to the assignment problem min $\sum_{i=1}^{n} \sum_{j=1} c_{ij} x_{ij}$ subject to constraints (4.3) to (4.5). The coefficients c_{ij} are defined as a function of distances and flows and the variables x_{ij} are binary. It is known that the assignment problem has the integrality property. Therefore a solution of it can be obtained by simply solving its linear programming relaxation. Along with the lower bound computation one gets the values of the dual variables u_i and v_i, $i = 1, \ldots, n$, corresponding to the constraints (4.3) and (4.4), respectively. The dual variables v_i are used to define a pre-ordering on the locations: the higher the value of the dual variable associated to a location, the higher the impact of the location on the QAP solution cost is assumed to be. Hence, the assignment of an object to that location is tried earlier.

The essential idea of ANTS is to use computations on the lower bounds on the completion cost of a partial solution in order to define the heuristic information about the attractiveness of adding a specific solution component (i, j) in (4.6). This is achieved by tentatively adding the solution component to the current partial solution and by estimating the cost of a complete solution, (containing that solution component), by means of a lower bound. This estimate is then used to influence the probabilistic decisions taken by the ant during the solution construction: the lower the estimate, the more attractive the addition of a specific pair. A further advantage of the usage of a lower bound is that an extension of a partial solution, for which the lower bound estimate is higher than the cost of the best solution found so far, can be discarded.

A disadvantage of the GLB, which is computed in an initialization phase of the algorithm, is that its computation is $\mathcal{O}(n^3)$. This is clearly expensive, since a lower bound has to be computed at each step during a solution construction of an ant. Therefore, Maniezzo proposes a lower bound weaker than GLB, the so-called LBD bound, which exploits the values of the dual variables to estimate the completion cost of a partial solution. Although LBD can be shown to be weaker than GLB, the main advantage of using LBD is its low computational complexity, which is $\mathcal{O}(n)$. For details on the lower bound computation we refer to [75]. Experimental results have shown that ANTS was at the time it was proposed one of the best available algorithms for the QAP. The good performance of the ANTS algorithm has been confirmed in a variety of further applications [77, 76].

4.4.2 Other Approaches

The integration of tree search techniques into constructive algorithms is an appealing possibility of hybridization. Given that ACO is, apart from GRASP, probably the most widespread used constructive metaheuristic, it is not astonishing that a number of other hybrids between ACO and tree search algorithms have been proposed. Probably the most noteworthy is the beam-ACO algorithm, which combines ACO with beam search, a derivative of branch-and-bound algorithms. Beam search is a tree search algorithm that keeps at each iteration a set of nodes of a search tree and expands each of them in several directions according to a selection based on lower bounds [88]. beam-ACO takes from ACO the probabilistic extension of partial solutions and uses a beam search type of management of partial solutions. It has shown very high performance for problems such as open shop scheduling [20] and assembly line balancing [21]. More details on beam-ACO and similar hybrid algorithms can be found in [22]. A combination of a GRASP algorithm with a branch-and-bound algorithm is described in [44] for an application to the JSP.

Another active area of hybridization is the integration of constraint programming techniques into constructive algorithms and, again, in particular into ACO. Such approaches have been described first by Meyer and Ernst [84] and in some more detail in a book chapter [83]. More recently, the integration of ACO into constraint programming languages has been considered [66].

4.5 Exploiting the Structure of Good Solutions

Many optimization problems show some type of structure in the sense that high quality solutions have a large number of solution components in common with optimal solutions. This observation is exploited by some hybrid methods. These define appropriate subproblems of the original problem by using the information obtained from high quality solutions. Often the resulting subproblems are small enough to be solved rather efficiently by exact algorithms.

This approach consists of two phases, which are executed in sequence. In the first phase, an approximate algorithm is used repeatedly to collect a number of high-quality solutions of the problem under consideration. Based on the solution components that are included in the set of solutions, a subproblem of the original problem is defined. It is expected that the subproblem still contains all or, if not all, most of the "important" decision variables and that the subproblem can be solved relatively easily by an exact algorithm. The optimal solution for the subproblem provides an upper bound for the optimal solution of the original problem.

An algorithmic outline of the type of methods falling into this framework is given next.

Algorithm 4.5.1 *Exploiting structure by collecting information*

```
Step 1: Initialization
        Let $\mathcal{I} = \emptyset$, where $\mathcal{I}$ is the set of collected solutions.
Step 2: Approximate algorithm
        while stopping_criterion is not met do
            Let $s$ be the solution returned after running an approximate
            algorithm.
            Add $s$ to $\mathcal{I}$.
        enddo
        Reduce $\mathcal{I}$ according to some criteria (optional).
Step 3: Optimization
            Define a subproblem $\mathcal{P}(\mathcal{I})$ that depends on $\mathcal{I}$.
            Find $opt(\mathcal{P}(\mathcal{I}))$, the optimal solution of $\mathcal{P}(\mathcal{I})$.
Step 4: Return $opt(\mathcal{P}(\mathcal{I}))$.
```

Here we discuss two well known examples where this idea was successfully applied to the p-median problem and the TSP.

4.5.1 Example: Heuristic Concentration

Rosing and ReVelle designed the heuristic concentration algorithm for the solution of the p-median problem [99]. Given a graph $G = (V, A)$, where $V = \{1, \ldots, n\}$, a weight associated with each node, and a distance associated with each arc, and given a positive integer p, the p-median problem consists of finding p nodes in the graph, called *facilities*, such that the sum of the weighted distances between every node and its closest facility is minimized. The nodes that are not facilities are called *demand nodes*.

The technique developed by Rosing and ReVelle is called *heuristic concentration*. In the first phase, a local search procedure, in this case the Teitz and Bart heuristic [105], is run repeatedly with different random starts. Every solution obtained in this process is recorded in a set $\mathcal{I}$. The number of times the heuristic is run is determined after conducting numerical experiments.

The second phase starts with the selection of a subset $\mathcal{I}'$ of the solutions in $\mathcal{I}$. $\mathcal{I}'$ contains only the best solutions of $\mathcal{I}$ with respect to the objective function. The number of solutions in $\mathcal{I}'$ is again determined by numerical experiments. The facility locations used in these “best” solutions form a subset of the set of all nodes. They will be collected in what the authors call the *concentration set* (CS). Finally, a restricted p-median problem, with the facility locations restricted to those contained in the concentration set, is solved.

Clearly, this method is built on the assumption that a near-optimal or optimal solution must have the facilities located at sites from CS. However, this is only an assumption based on experimental results. An even stronger assumption is made by the authors in order to further reduce the dimension of the subproblem being solved. This is the claim that the nodes that are facilities in *all* solutions considered *must* be facilities. The rest of the CS contains *possible* locations. Therefore, they split CS into two subsets: one contains the locations where there are facilities in all solutions, called *CS open* (CS_0), and the other one contains the nodes where there is a facility in at least one but not all solutions. The latter subset of CS is called *CS free* (CS_f). The authors exploit the fact that each demand node not in CS must be assigned to the closest facility. For every node, variables are needed only for that node in CS_0 that is closest to the demand node, and only for those nodes in CS_f that are even closer than that member of the CS_0.

It is clear that when CS can be split, the size of the second subproblem is smaller than the size of the first one. However, there is no guarantee that such a partition of CS exists. When $CS_0 = \emptyset$, the only possibility is to use the first subproblem. Since in most cases the solution of the LP relaxation of the binary integer model of a p-median problem is integer, Rosing and ReVelle solve the LP relaxation of the restricted problem using an LP solver (in this case CPLEX). Even when the solution is fractional, an integer solution is found very easily. Numerical results are provided in both [99] and subsequent papers [98, 100]. They prove that the heuristic concentration is a viable technique that obtains very good results in practice.

4.5.2 Example: Tour Merging

Applegate et al. [9] proposed the tour merging approach, which was further refined by Cook and Seymour [36]. Tour merging is a two phase procedure following the steps outlined above.

The first phase consists of generating a number of high quality tours for the TSP instance that needs to be solved. While Applegate et al. [9] used their chained Lin-Kernighan (LK) algorithm, in the later study in [36] the iterated version of Helsgaun's LK implementation [60] is used. This heuristic reaches significantly better quality solutions and using it to define the set of tours $\mathcal{I}$ in the first stage was found to result in better quality tours in the second stage. Additionally, given that for the same number of tours the number of decision variables is smaller, larger instances can be tackled in this way.

The second phase consists of solving the TSP on the graph induced by the set of tours $\mathcal{I}$ on the original graph. In other words, a TSP is solved on a restricted graph that has the same set of nodes as the original graph, but its set of edges consists of the edges that appear at least once in any of the tours in $\mathcal{I}$. Formally, if the original graph is $G = (V, A)$, where V is the set of nodes and A the set of arcs, the reduced graph can be described as $G' = (V, A')$, where $A' = \{a \in A : \exists t \in \mathcal{I}, a \in t\}$.

Several possibilities of solving the TSP on the reduced graph G' can be considered. In [9], an exact solver part of the Concorde package, was used. When Concorde was applied to medium sized instances (with up to 5000 cities), the tour merging method could identify optimal solutions for all instances at least twice out of ten independent trials. However, on few instances the computation time was large, with most of it being taken by the optimization code applied in the second stage. Another possibility of solving the TSP on G' is to use special purpose algorithms that exploit the structure of the graph G'. In particular, G' is a very sparse graph and it is likely to have a low *branch-width* [97]. In [36], a dynamic programming algorithm that exploits this property is presented and computational results on a wide range of large TSP instances (2,000 up to about 20,000 cities) show excellent performance, resulting in computation times for the second stage that are much shorter than the times required by a general purpose TSP solver. In fact, the tour merging approach is currently one of the best performing approximate algorithms for medium to large TSP instances.

4.5.3 Discussion

The two examples show that approaches based on collected information can result in highly efficient hybrid algorithms in practice. For the approach to work on specific problems, high-quality solutions need to have specific characteristics. Firstly, it is important that these solutions have many components in common. Then the resulting subproblem will have a small number of decision variables and will be amenable to solving with exact algorithms. Secondly, the resulting subproblem is required to contain most of (ideally, all) the important decision variables to ensure that the second stage can return a very high quality solution. Such properties appear to hold in the TSP case: it is a well known property that local minima cluster in a small region of the search space and that the better their quality, the more edges they have in common with an optimal solution [23, 67, 82]. Some evidence exists that similar conclusions hold for the p-median problem [57]. Notions of search space analysis are important in this context. Intuitively, problems with a high *fitness distance correlation* are more likely to be amenable for such algorithmic approaches. Roughly speaking, a high fitness distance correlation indicates that the better the solutions the closer they are to an optimal solution in terms of an appropriate distance definition.

Clearly, the ideas outlined in this section are not necessarily restricted to using an exact algorithm in the second stage. For example, in the case of the set covering problem (SCP), Finger et al. [45] first provide a fitness distance correlation analysis of the SCP search space. Based on the result that for some classes of SCP instances the characteristics mentioned above were satisfied, a subproblem is defined by the solutions returned by several local searches and then solved by simulated annealing [25]. Puchinger et al. [95] study the "core" concept [16, 92] for the multidimensional knapsack problem and apply SLS algorithms (a memetic algorithm and a relaxation guided variable neighborhood search) and exact algorithms to the resulting reduced instances. They obtain very high quality solutions and show that the SLS algorithms reach better quality solutions on the "core" instances than when applying them directly to the original instances. (Note that the concept of "cores" is often defined through linear relaxations, e.g. by fixing variables that are integer in the relaxation; these approaches are related to those that are discussed in Section 4.6.)

Finally, it should be mentioned that the approaches described in this section are related to some already discussed in Section 4.3. The "optimized crossover" algorithms described there define a new problem similar to the union of solution components in tour merging or heuristic concentration. In fact, the latter two could be seen as an "optimized multi-parent recombination" operator. A difference is that the approaches presented here were originally conceived as consisting of two stages that are executed in sequence. However, alternative and intermediate approaches can easily be conceived.

4.6 Exploiting Information from Relaxations in Metaheuristics

A more frequent way of combining elements from exact algorithms and metaheuristics is the exploitation of some type of information obtained from a relaxation of the original problem to bias in some way the SLS algorithm. Various possibilities have been explored from rather straightforward uses to more complex ones. From the latter ones we present an example given by Vasquez and Hao [111] for the multidimensional knapsack problem. We then give some pointers to more literature on the exploitation of relaxations.

4.6.1 Example: Simplex and Tabu Search Hybrid

Vasquez and Hao [111] combine a tabu search with an exact method, in this case a simplex algorithm that solves some linear programming subproblems. Given m resources, each having a corresponding resource limit, n objects, each with an associated profit, and an associated vector of resource consumption, the 0-1 multidimensional knapsack problem seeks to maximize the total profit made by using the objects such that the amount of resources consumed is within the resource limits. In what follows we will denote the profit associated with object i by p_i, the amount of resource j consumed by selecting object i by r_{ji}, and the limit of resource j by R_j. The standard integer programming formulation of the 0-1 multidimensional knapsack problem can be written as:

$$\begin{aligned} \max \ & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{n} r_{ji} x_i \le R_j, \quad \forall j = 1, \ldots, m && (4.7) \\ & x_i \in \{0,1\}, \qquad\quad \forall i = 1, \ldots, n, && (4.8) \end{aligned}$$

where $x_i = 1$ if object i is used and $x_i = 0$ otherwise.

The method starts by relaxing the integrality constraint. Since the solution of the linear programming relaxation of an integer programming problem can be far away from the integer optimal solution, the linear programming relaxation is then “strengthened” by the addition of an extra constraint. It is clear that the integer optimal solution of the 0-1 multidimensional knapsack problem can have only a certain number of components that are *not* zero. Based on this observation, the constraint added to the linear programming relaxation enforces the number of non-zero components to be equal to k, where $0 \le k \le n$ (see constraint (4.9) below). Therefore $n + 1$ linear programming problems are obtained (one for each value of k):

$$\begin{aligned} \max \ & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{n} r_{ji} x_i \le R_j, \quad \forall j = 1, \ldots, m \\ & \sum_{i=1}^{n} x_i = k && (4.9) \\ & x_i \in [0,1], \qquad\quad \forall i = 1, \ldots, n && (4.10) \end{aligned}$$

We denote such a subproblem by $\mathcal{P}(k)$. Clearly, the solutions of these problems can be fractional. However, it is hoped that the optimal solution of the original problem is close to one of the optimal solutions obtained after solving the relaxed problems. (To reduce the number of subproblems that need to be solved, Vasquez and Hao compute bounds on k; for details see [111].)

After the reduction, each linear program is solved by the simplex algorithm. The solutions returned are then used to generate starting solutions to a following tabu search phase. More precisely, an initial solution is obtained from a possibly fractional solution $x^{[k]}$ to $\mathcal{P}(k)$ by fixing its largest k elements to one and the rest to zero. The solution $x^{[k]}$ of $\mathcal{P}(k)$ has the additional use to restrict the search space explored by a tabu search procedure. In fact, the subsequently run tabu search algorithm, which is based on the reverse elimination method [50], is restricted to explore only solutions x for which it holds that $\sum_{i=1}^{n} |x_i - x_i^{[k]}| \leq \delta_{max}$, i.e. the solutions explored by the tabu search are limited to a ball of radius δ_{max} around $x^{[k]}$. The radius δ_{max} is computed heuristically as follows. Let no_int denote the number of integer components of $x^{[k]}$ and no_frac the number of fractional components; then, the radius is set to a value of $\delta_{max} \approx 2 \times (no\_int + no\_frac - k)$. Note that $\delta_{max} = 0$ corresponds to the case when an integer solution is returned by simplex. The search space is further reduced by limiting the number of objects taken into configurations. For each k, the number of non-zero components of a candidate solution is considered to be exactly k and only the *integer* candidate solutions are considered. This actually corresponds to a partitioning of the search space. The neighborhoods are defined as add/drop neighborhoods. The neighborhood of a given configuration is the set of configurations that differ by exactly two elements from the initial configuration, while keeping the number of non-zero elements constant.

An extensive computational study of the final algorithm is provided. The authors report many improved results compared to the best known results at the time of publication; however, they acknowledge large computational time of about three days on a set of computers with Pentium 300MHz or 500 MHz computers for very large instances with as many as 2,500 items and 100 constraints. Further improvements on this approach are described in [112].

4.6.2 Discussion

The paper of Hao and Vasquez is an example of how information of solutions of relaxations of IP formulations can be used to restrict the area of the search space examined by local search and to provide good initial solutions for the local search. As said before, there are a number of other approaches that in some form make use of relaxations and information derived from them. How relaxations and problem decompositions can be used to define heuristic methods and metaheuristics is shown in Chapter 5 of this book.

A particularly active area appears to be the exploitation of lower bound computations through Lagrangean relaxation (LR). (For an introduction to Lagrangean relaxation and its use in combinatorial optimization we refer to [18].) Such approaches have been proposed for a variety of problems and for several they result in state-of-the-art algorithms. The central idea in these approaches is to use Lagrangean multipliers and costs to guide construction or local search type heuristics. Among the first approaches that use LR is the heuristic of Beasley [17]. Currently, state-of-the-art algorithms for the set covering problem are, at least in part, based on exploiting information from LR [29, 32]. Ways of integrating information gained in LR into a tabu search algorithm is given in [56] and computational results are presented for an example application to the capacitated warehouse location problem. Other, recent examples of using LR to direct SLS algorithms include a genetic algorithm for the prize collecting Steiner tree problem [59], exploiting Lagrangean decomposition for an evolutionary algorithm for the knapsack constrained maximum spanning tree problem [91], or hybrid approaches to the design of the last mile in fiber optic networks [69].

4.7 Conclusions

In this chapter we reviewed existing approaches that combine stochastic local search and methods from the area of integer programming for the solution of $\mathcal{NP}$-hard combinatorial optimization problems, extending significantly over our earlier review [41]. We focused on techniques where the "master" routine is based on some stochastic local search method and strengthened by the use of exact algorithms; see Chapter 3 for the case where the IP method is the master. In the terms used in [96], the discussed approaches can be classified as integrative or as collaborative, sequential approaches. In the former class (covered here in Sections 4.2 – 4.4), the exact algorithms are used to repeatedly solve subproblems that arise in the SLS algorithm. In the latter type of approaches (discussed in Sections 4.5 and 4.6), an exact methods is executed either before the start of the SLS part, typically, to produce information that is later used to bias the local search, or used as a post-processing facility to improve solutions generated by typically several local search processes.

The main conclusion we can make here is that there are many research opportunities to develop algorithms that integrate local search and exact techniques. Despite the fact that local search and exact algorithms have somewhat complementary advantages and disadvantages, relatively few researches present algorithms that try to combine the two areas. One reason for this may be that such combined methods are often rather complex and hence they may involve significantly long development times. A possibly more important obstacle is that they require strong knowledge about the techniques available in two rather different techniques, which are often treated in different research streams. Fortunately, this situation appears to be improving, as exemplified by the contributions in this book. In general, we strongly believe that combinations of exact methods and stochastic local search methods are a very promising direction for future research in combinatorial optimization.

Acknowledgements This work has been supported by META-X, an ARC project funded by the French Community of Belgium. TS acknowledges support from the fund for scientific research F.R.S.-FNRS of the French Community of Belgium, of which he is a research associate.

References

1. E.H.L. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, Chichester, 1997.
2. C.C. Aggarwal, J.B. Orlin, and R.P. Tai. An optimized crossover for the maximum independent set. *Operations Research*, 45:226–234, 1997.
3. R.K. Ahuja, Ö.Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1–3):75–102, 2002.
4. R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.P. Punnen. Very large-scale neighborhood search: Theory, algorithms, and applications. In T.F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, pages 20–1—20–15. Chapman & Hall/CRC, Boca Raton, FL, 2007.
5. R.K. Ahuja, J.B. Orlin, and D. Sharma. Multi-exchange neighbourhood structures for the capacitated minimum spaning tree problem. Working Paper, 2000.
6. R.K. Ahuja, J.B. Orlin, and D. Sharma. Very large-scale neighbourhood search. *International Transactions in Operational Research*, 7(4–5):301–317, 2000.
7. R.K. Ahuja, J.B. Orlin, and D. Sharma. A composite very large-scale neighborhood structure for the capacitated minimum spanning tree problem. *Operations Research Letters*, 31(3):185–194, 2003.
8. E. Angel and E. Bampis. A multi-start dynasearch algorithm for the time dependent single-machine total weighted tardiness scheduling problem. *European Journal of Operational Research*, 162(1):281–289, 2005.
9. D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Finding Tours in the TSP. Technical Report 99885, Forschungsinstitut für Diskrete Mathematik, University of Bonn, Germany, 1999.
10. D. Applegate, R.E. Bixby, V. Chvátal, and W. Cook. Concorde TSP solver. `http://www.tsp.gatech.edu//concorde`, last visited December 2008.
11. D. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, Princeton, NJ, 2006.
12. D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.
13. E. Balas and W. Niehaus. Finding large cliques in arbitrary graphs by bipartite matching. In D. S. Johnson and M. A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, pages 29–53. American Mathematical Society, 1996.
14. E. Balas and W. Niehaus. Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems. *Journal of Heuristics*, 4(2):107–122, 1998.
15. E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.
16. E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):1130–1154, 1980.
17. J.E. Beasley. A Lagrangian heuristic for set covering problems. *Naval Research Logistics*, 37:151–164, 1990.
18. J.E. Beasley. Lagrangean relaxation. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. Blackwell Scientific Publications, 1993.
19. J.L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992.
20. C. Blum. Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

21. C. Blum. Beam-ACO for simple assembly line balancing. *INFORMS Journal on Computing*, 20(4):618–627, 2008.
22. C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, and M. Mastrolilli. Hybridizations of metaheuristics with branch & bound derivatives. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 85–116. Springer Verlag, Berlin, 2008.
23. K.D. Boese, A.B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimization. *Operations Research Letters*, 16:101–113, 1994.
24. P. Borisovsky, A. Dolgui, and A. Eremeev. Genetic algorithms for a supply management problem: MIP-recombination vs. greedy decoder. *European Journal of Operational Research*, 195(3):770–779, 2009.
25. M.J. Brusco, L.W. Jacobs, and G.M. Thompson. A morphing procedure to supplement a simulated annealing heuristic for cost- and coverage-correlated set covering problems. *Annals of Operations Research*, 86:611–627, 1999.
26. K. Büdenbender, T. Grünert, and H.-J. Sebastian. A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem. *Transportation Science*, 34(4):364–380, 2000.
27. E.K. Burke, P. Cowling, and R. Keuthen. Embedded local search and variable neighbourhood search heuristics applied to the travelling salesman problem. Technical report, University of Nottingham, 2000.
28. E.K. Burke, P.I. Cowling, and R. Keuthen. Effective local and guided variable neighbourhood search methods for the asymmetric travelling salesman problem. In E.J.W. Boers, J. Gottlieb, P.L. Lanzi, R.E. Smith, S. Cagnoni, E. Hart, G.R. Raidl, and H. Tijink, editors, *Applications of Evolutionary Computing*, volume 2037 of *Lecture Notes in Computer Science*, pages 203–212. Springer Verlag, Berlin, 2001.
29. A. Caprara, M. Fischetti, and P. Toth. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.
30. J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.
31. Y. Caseau and F. Laburthe. Disjunctive scheduling with task intervals. Technical Report LIENS 95-25, Ecole Normale Superieure Paris, France, July 1995.
32. S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228, 1995.
33. M. Chiarandini, I. Dumitrescu, and T. Stützle. Very large-scale neighborhood search: Overview and case studies on coloring problems. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 117–150. Springer Verlag, Berlin, 2008.
34. R.K. Congram. *Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimization.* PhD thesis, Southampton University, Faculty of Mathematical Studies, Southampton, UK, 2000.
35. R.K. Congram, C.N. Potts, and S.L. Van de Velde. An iterated dynasearch algorithm for the single–machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.
36. W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.
37. P.I. Cowling and R. Keuthen. Embedded local search approaches for routing optimization. *Computers & Operations Research*, 32(3):465–490, 2005.
38. M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications and advances. In Glover and Kochenberger [52], pages 251–285.
39. M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, 2004.
40. I. Dumitrescu. *Constrained Shortest Path and Cycle Problems.* PhD thesis, The University of Melbourne, 2002.

41. I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. In G.R. Raidl, J.A. Meyer, M. Middendorf, S. Cagnoni, J.J.R. Cardalda, D.W. Corne, J. Gottlieb, A. Guillot, E. Hart, C.G. Johnson, and E. Marchiori, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 211–223. Springer Verlag, Berlin, 2003.
42. A.V. Eremeev. On complexity of optimal recombination for binary representations of solutions. *Evolutionary Computation*, 16(1):127–147, 2008.
43. S. Fernandes and H.R. Lourenço. Optimised search heuristic combining valid inequalities and tabu search. In M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics, 5th International Workshop, HM 2008*, volume 5296 of *Lecture Notes in Computer Science*, pages 87–101. Springer Verlag, Berlin, 2008.
44. S. Fernandes and H.R. Lourençou. A simple optimised search heuristic for the job shop scheduling problem. In C. Cotta and J.I. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 203–218. Springer Verlag, Berlin, 2008.
45. M. Finger, T. Stützle, and H.R. Lourençou. Exploiting fitness distance correlation of set covering problems. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G.R. Raidl, editors, *Applications of Evolutionary Computing*, volume 2279 of *Lecture Notes in Computer Science*, pages 61–71. Springer Verlag, Berlin, 2002.
46. M. Fischetti and A. Lodi. Local branching. *Mathematical Programming, Series B*, 98:23–47, 2003.
47. F. Focacci, F. Laburthe, and A. Lodi. Local search and constraint programming. In Glover and Kochenberger [52], pages 369–403.
48. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness.* Freeman, San Francisco, CA, 1979.
49. P.C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the SIAM*, 10:305–313, 1962.
50. F. Glover. Tabu search. *ORSA Journal on Computing*, 2:4–32, 1990.
51. F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65:223–253, 1996.
52. F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2002.
53. F. Glover, M. Laguna, and R. Martí. Scatter search and path relinking: Advances and applications. In Glover and Kochenberger [52], pages 1–35.
54. GLPK (GNU Linear Programming Kit). `http://www.gnu.org/software/glpk/glpk.html`, last visited December 2008.
55. A. Grosso, F. Della Croce, and R. Tadei. An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32(1):68–72, 2004.
56. T. Grünert. Lagrangean tabu search. In P. Hansen and C.C. Ribeiro, editors, *Essays and Surveys on Metaheuristics*, pages 379–397. Kluwer Academic Publishers, Boston, MA, 2002.
57. P. Hansen and N. Mladenoviç. Variable neighbourhood search for the p-median. *Location Science*, 5(4):207–226, 1998.
58. P. Hansen, N. Mladenovic, and D. Perez-Britos. Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350, 2001.
59. M. Haouari and J.C. Siala. A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Computers & Operations Research*, 33(5):1274–1288, 2006.
60. K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.
61. H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann Publishers, San Francisco, CA, 2005.

62. B. Hu, M. Leitner, and G.R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, pages 473–499, 2008.
63. ILOG. `http://www.ilog.com/products/cplex/`, 2008.
64. D.S. Johnson and L.A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, 2002.
65. B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technology Journal*, 49:213–219, 1970.
66. M. Khichane, P. Albert, and C. Solnon. Integration of ACO in a constraint programming language. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle, and A.F.T. Winfield, editors, *Ant Colony Optimization and Swarm Intelligence, 6th International Conference, ANTS 2008*, volume 5217 of *Lecture Notes in Computer Science*, pages 84–95. Springer Verlag, Berlin, 2008.
67. S. Kirkpatrick and G. Toulouse. Configuration space analysis of travelling salesman problems. *Journal de Physique*, 46(8):1277–1292, 1985.
68. E.L. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963.
69. M. Leitner and G.R. Raidl. Lagrangian decomposition, metaheuristics, and hybrid approaches for the design of the last mile in fiber optic networks. In M.J. Blesa, C. Blum, C. Cotta, A.J. Fernández, J.E. Gallardo, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics, 5th International Workshop, HM 2008*, volume 5296 of *Lecture Notes in Computer Science*, pages 158–174. Springer Verlag, Berlin, 2008.
70. S. Lin and B.W. Kernighan. An effective heuristic algorithm for the travelling salesman problem. *Operations Research*, 21:498–516, 1973.
71. H.R. Lourenço. *A Computational Study of the Job-Shop and the Flow-Shop Scheduling Problems*. PhD thesis, School of Or & IE, Cornell University, Ithaca, NY, 1993.
72. H.R. Lourenço. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research*, 83:347–367, 1995.
73. H.R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In Glover and Kochenberger [52], pages 321–353.
74. H.R. Lourenço, J.P. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35(3):331–343, 2001.
75. V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
76. V. Maniezzo and A. Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8):927–935, 2000.
77. V. Maniezzo, A. Carbonaro, M. Golfarelli, and S. Rizzi. An ANTS algorithm for optimizing the materialization of fragmented views in data warehouses: Preliminary results. In *Applications of Evolutionary Computing, EvoWorkshops 2001*, volume 2037 of *Lecture Notes in Computer Science*, pages 80–89. Springer Verlag, Berlin, 2001.
78. K. Marriott and P. Stuckey. *Programming with Constraints.* MIT Press, Cambridge, MA, 1998.
79. T. Mautor. Intensification neighbourhoods for local search methods. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 493–508. Kluwer Academic Publishers, Norwell, MA, 2002.
80. T. Mautor and P. Michelon. MIMAUSA: A new hybrid method combining exact solution and local search. In *Extended abstracts of the 2nd International Conference on Meta-heuristics*, page 15, Sophia-Antipolis, France, 1997.
81. T. Mautor and P. Michelon. MIMAUSA: an application of referent domain optimization. Technical Report 260, Laboratoire d'Informatique, Université d'Avignon et des Pays de Vaucluse, Avignon, France, 2001.

82. P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 244–260. McGraw Hill, London, UK, 1999.
83. B. Meyer. Hybrids of constructive metaheuristics and constraint programming. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 85–116. Springer Verlag, Berlin, 2008.
84. B. Meyer and A. Ernst. Integrating ACO and constraint propagation. In M. Dorigo, M. Birattari, C. Blum, L.M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, volume 3172 of *Lecture Notes in Computer Science*, pages 166–177. Springer Verlag, Berlin, 2004.
85. MINTO - Mixed INTeger Optimizer. `http://coral.ie.lehigh.edu/minto`, last visited December 2008.
86. G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
87. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job-shop problem. *Management Science*, 42(2):797–813, 1996.
88. P.S. Ow and T.E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.
89. G. Pesant and M. Gendreau. A view of local search in constraint programming. In E. Freuder, editor, *Proceedings of Constraint Programming 1996*, volume 1118 of *Lecture Notes in Computer Science*, pages 353–366. Springer Verlag, Berlin, 1996.
90. G. Pesant and M. Gendreau. A constraint programming framework for local search methods. *Journal of Heuristics*, 5:255–279, 1999.
91. S. Pirkwieser, G.R. Raidl, and J. Puchinger. A Lagrangian decomposition / evolutionary algorithm hybrid for the knapsack constrained maximum spanning tree problem. In C. Cotta and J.I. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 69–85. Springer Verlag, Berlin, 2008.
92. D. Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47(4):570–575, 1999.
93. C.N. Potts and S. van de Velde. Dynasearch: Iterative local improvement by dynamic programming; part I, the traveling salesman problem. Technical Report LPOM–9511, Faculty of Mechanical Engineering, University of Twente, Enschede, The Netherlands, 1995.
94. M. Prandtstetter and G.R. Raidl. An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *European Journal of Operational Research*, 191(3):1004–1022, 2008.
95. J. Puchinger, G.R. Raidl, and U. Pferschy. The core concept for the multidimensional knapsack problem. In J. Gottlieb and G.R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization–EvoCOP 2006*, volume 3906 of *Lecture Notes in Computer Science*, pages 195–208. Springer Verlag, Berlin, 2006.
96. G.R. Raidl and J. Puchinger. Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M.J. Blesa, A. Roli, and M. Sampels, editors, *Hybrid Metaheuristics—An Emergent Approach to Optimization*, volume 117 of *Studies in Computational Intelligence*, pages 31–62. Springer Verlag, Berlin, 2008.
97. N. Robertson and P.D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory*, 52:153–190, 1991.
98. K.E. Rosing. Heuristic concentration: a study of stage one. *Environment and Planning B: Planning and Design*, 27(1):137–150, 2000.
99. K.E. Rosing and C.S. ReVelle. Heuristic concentration: Two stage solution construction. *European Journal of Operational Research*, pages 955–961, 1997.

100. K.E. Rosing and C.S. ReVelle. Heuristic concentration and tabu search: A head to head comparison. *European Journal of Operational Research*, 117(3):522–532, 1998.
101. B. Roy and B. Sussmann. Les problemes d'ordonnancement avec constraintes disjonctives. Notes DS no. 9 bis, SEMA.
102. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming - CP98, 4th International Conference*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer Verlag, Berlin, 1998.
103. M. Sniedovich and S. Voß. The corridor method: A dynamic programming inspired metaheuristic. *Control and Cybernetics*, 35:551–578, 2006.
104. É.D. Taillard and S. Voß. POPMUSIC: Partial optimization metaheuristic under special intensification conditions. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in metaheuristics*, pages 613–629. Kluwer Academic Publishers, Boston, MA, 2002.
105. M.B. Teitz and P. Bart. Heuristic methods for estimating the generalized vertex median of a weighted graph. *Operations Research*, 16:955–961, 1968.
106. P.M. Thompson and J.B. Orlin. The theory of cycle transfers. Working Paper No. OR 200-89, 1989.
107. P.M. Thompson and H.N. Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.
108. P. Toth and D. Vigo, editors. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2002.
109. P. van Hentenryck. *The OPL Optimization Programming Language.* MIT Press, Cambridge, MA, 1999.
110. P. Van Hentenryck and L. Michel. *Constraint-Based Local Search.* MIT Press, Cambridge, MA, 2005.
111. M. Vasquez and J.-K. Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In B. Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 328–333. Morgan Kaufmann Publishers, San Francisco, CA, 2001.
112. M. Vasquez and Y. Vimont. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1):70–81, 2005.
113. Xpress-MP. `http://www.dashoptimization.com/home//products/products_optimizer.html`, last visited December 2008.
114. M. Yagiura and T. Ibaraki. The use of dynamic programming in genetic algorithms for permutation problems. *European Journal of Operational Research*, 92:387–401, 1996.